intel®

AP-428

APPLICATION NOTE

# Distributed Motor Control Using the 80C196KB

**TIM SCHAFER**
**MICHAEL CHEVALIER**
80C196KB APPLICATIONS

December 1993

Order Number: 270701-001

Information in this document is provided in connection with Intel products. Intel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Intel products except as provided in Intel's Terms and Conditions of Sale for such products.

Intel retains the right to make changes to these specifications at any time, without notice. Microcomputer Products may have minor variations to this specification known as errata.

*Other brands and names are the property of their respective owners.

†Since publication of documents referenced in this document, registration of the Pentium, OverDrive and iCOMP trademarks has been issued to Intel Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

# DISTRIBUTED MOTOR CONTROL USING THE 80C196KB

## CONTENTS

## 1.0 INTRODUCTION

Distributed control of servo motors has a wide range of applications including industrial control, factory automation and robotics. The tasks involved in controlling a servo motor include position and velocity measurement, implementation of control algorithms, detection of overrun and stress conditions, and communication back to a central controller. The 80C196KB high performance microcontroller provides a low cost solution for handling these required control tasks.

The 80C196KB microcontroller is a highly integrated and high performance member of the MCS®-96 family. The part is available in ROM (83C196KB) and EPROM (87C196KB) versions. A block diagram of the 80C196KB is shown in Figure 2. The availability of a variety of on board peripherals such as timer/counters, A/D, PWM, Serial Port and High Speed Input and Output capture/compare timer subsystem provides for a flexible architecture for control applications at a reasonable cost.

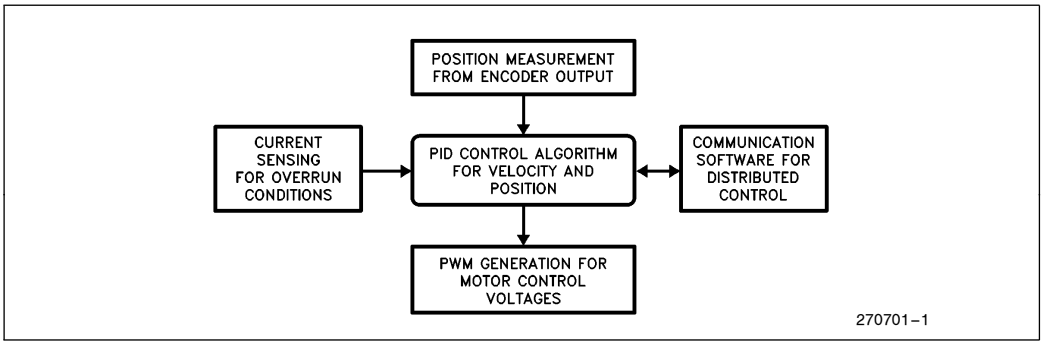



**Figure 1. Control Tasks for Distributed Control of a Servo Motor**

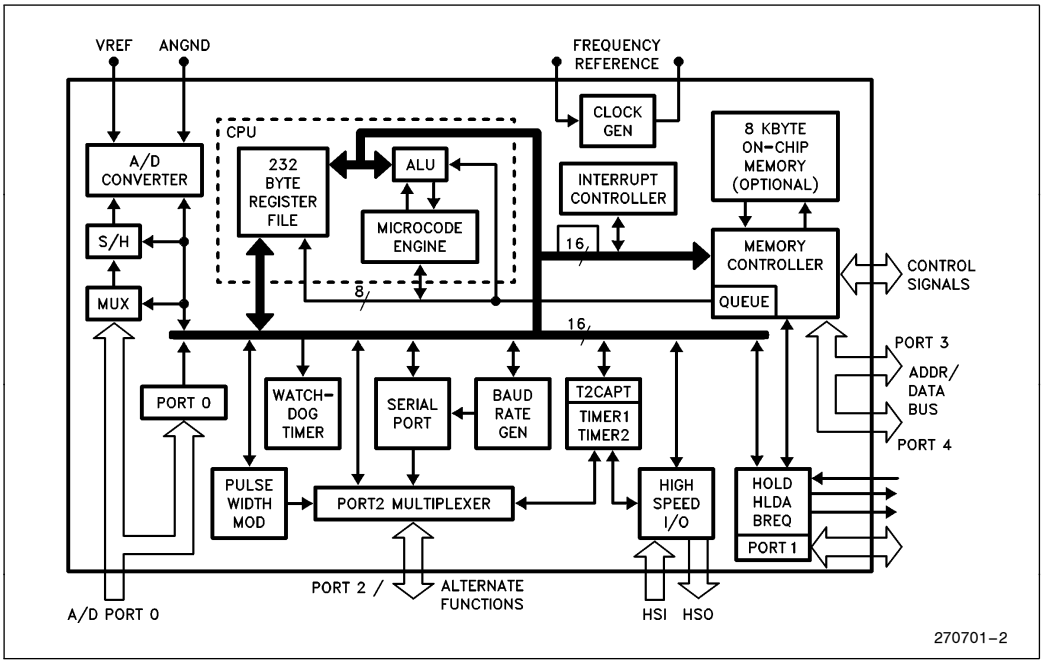



**Figure 2. 80C196KB Block Diagram**

This application note describes several different methods for motor control using the on board peripherals of the 80C196KB. Hardware and Software techniques are addressed to generate PWMs for driving motors and to measure position from the output of precision optical encoders.

A Proportional, Integral and Differential (PID) algorithm controls both the position and velocity of the motor. The PID algorithm employs proportional, integral and differential feedback to control the system characteristics of the motor. The motor can be moved either manually or under the control of a velocity profile. The mode used to position the motor is determined by commands received from a master controller.

Communication to the master controller was implemented using the onboard serial port of the 80C196KB. The application of distributed control to position and program a six axis robot arm using six separate motors will be described. Each 80C196KB motor controller acts as a slave under control of the master. An IBM PC was selected as the master controller for the robot. Turbo Prolog was used to develop the human interface. A robot programming language and control screen was produced to program movements of each individual motor.

The motor control hardware, taking full advantage of the peripheral features of the 80C196KB, will be discussed first. The control software will be discussed later.

## 2.0 HARDWARE

The hardware tasks required to control a servo motor under the command of a centralized controller include the following:

1) Feedback of the motor position and direction.

2) Control of the motor speed and direction.

3) Detection of motor overrun conditions.

4) Communication from/to a master controller.

Two different hardware interface examples for controlling a servo motor are shown in Figures 3 and 4. The first example controls one motor using TIMER2 and the dedicated PWM unit on the 80C196KB and would best fit a high performance, high resolution application. Example number 2 uses the HSI (High Speed Inputs) and the HSO (High Speed Outputs) to control two motors. The second method can control up to four motors by trading off some performance and resolution.

This section deals with the hardware and software requirements of acquiring position feedback from incremental shaft encoders and generating outputs to drive DC servo motors. A current limiting circuit which is useful in determining when the motor has stalled is also presented. Current monitoring can also control the torque to prevent the motor from crushing an object. The closed loop digital control algorithms are discussed in the software section.

### 2.1 Optical Encoders

Optical encoders can be used to measure the position of rotating equipment. They provide a cost effective solution for digital position and velocity feedback to a microcontroller. Encoders produce two pulse trains which give an incremental position count. Velocity and acceleration may be calculated by measuring the number of counts in a given sample period. Or, in a slow speed system, velocity and acceleration can be measured directly from the time between edges of the pulse train. Acceleration and velocity calculations are discussed in detail in the software section.

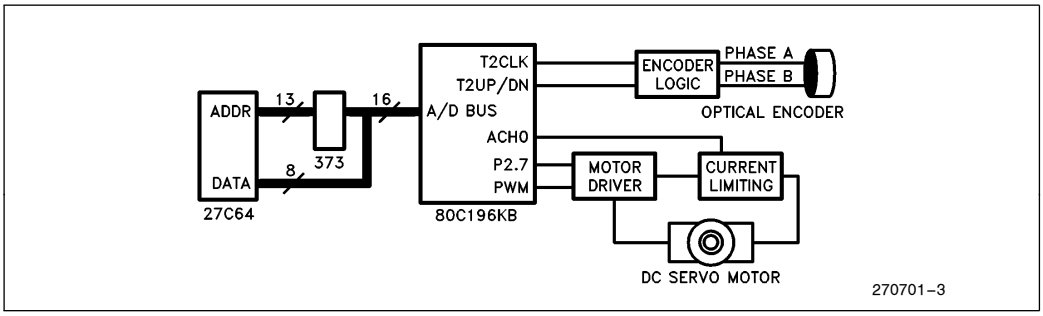



**Figure 3. Block Diagram of Motor Control Hardware using PWM and TIMER2**

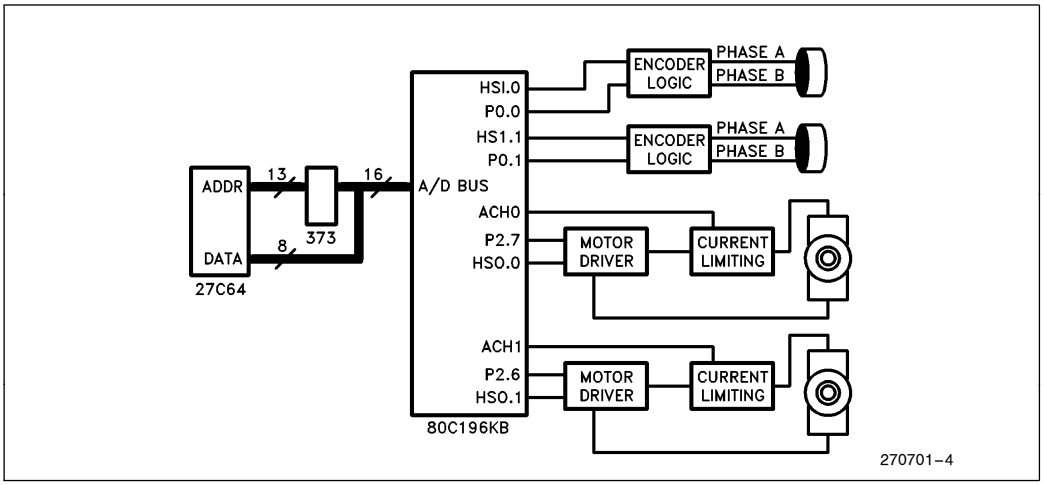



**Figure 4. Block Diagram of Motor Control Hardware using HSO and HSI**

Pulse trains from an encoder can vary from two pulses per revolution for low cost applications, to over 5000 pulses per revolution for high resolution requirements. Figure 5 shows an eight line encoder along with the associated waveforms. A small amount of external logic and a few discrete components decode a position count and a direction indication from phase A and phase B.

External logic for encoders is shown in Figure 6. Figure 7 shows a timing diagram of the circuit. Bold type denotes the input and desired output waveforms. The phases from an encoder are mechanically produced electrical signals. When the motor rotates slowly, the phases inherently exhibit slow rise and fall times. The four Schmitt triggers in the circuit protect against oscillation in the digital circuit due to these long rise and fall times.

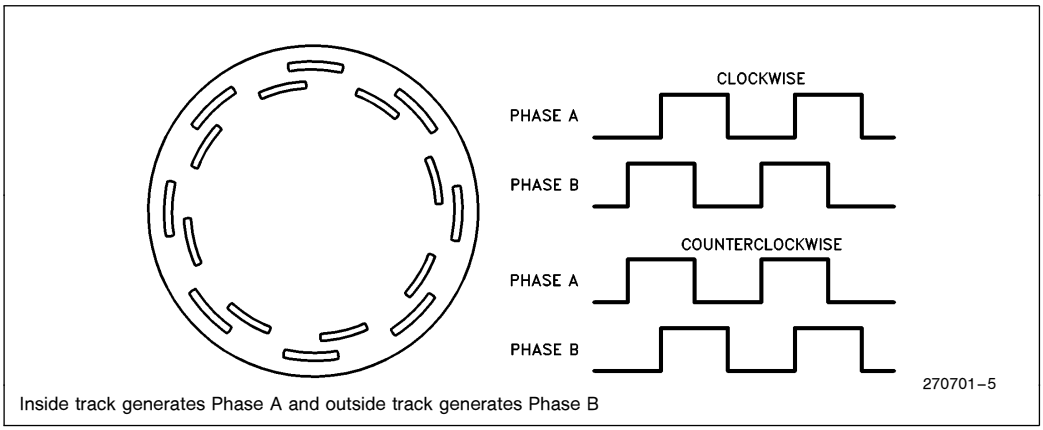



**Figure 5. Eight Line Encoder**

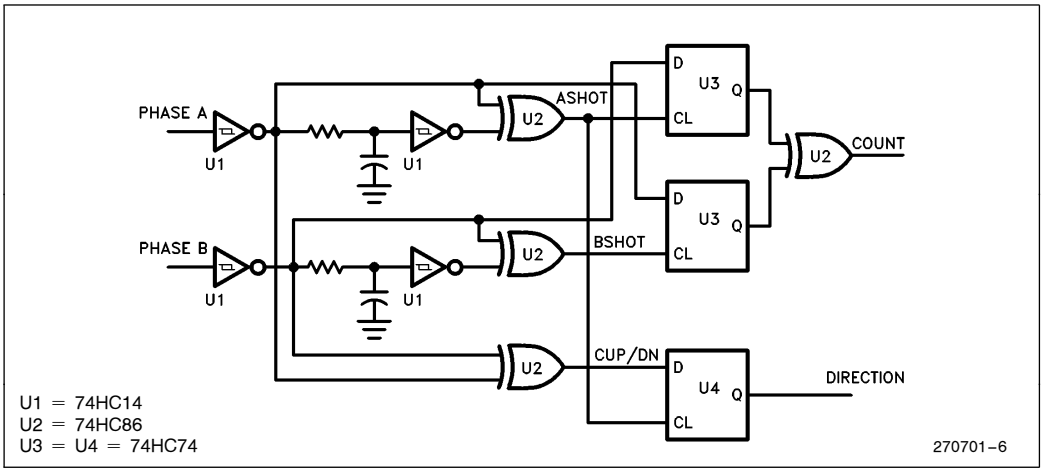



**Figure 6. External Logic for Encoders**

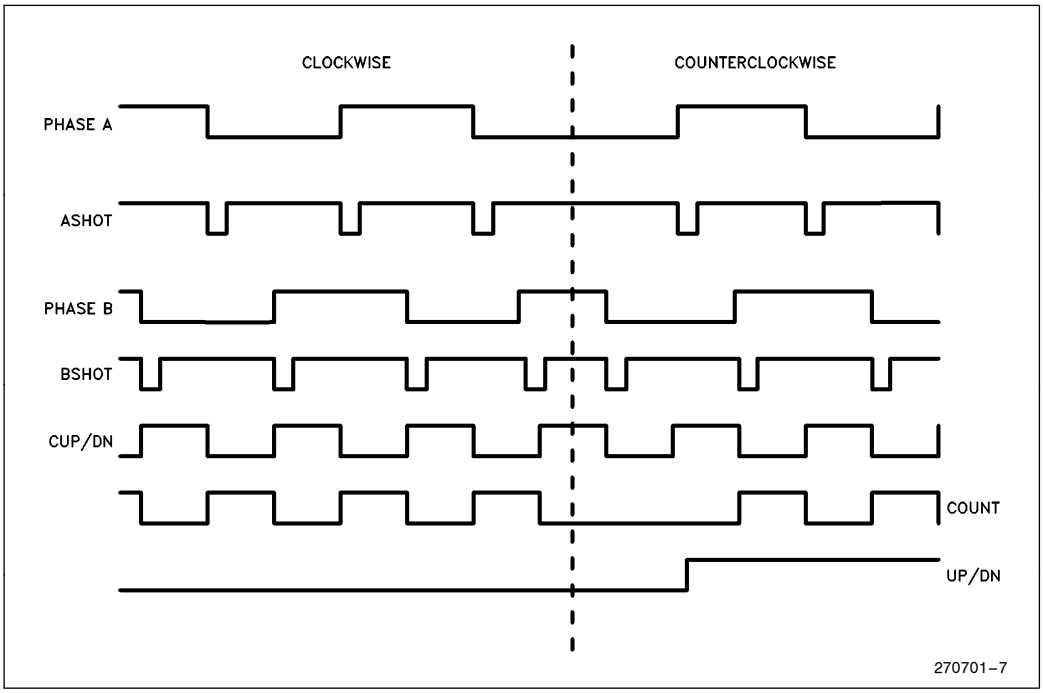



**Figure 7. Timing Diagram for Logic Circuit**

A simple one-shot is constructed with an RC circuit and an XOR gate to generate a pulse on each edge of each phase. ASHOT clocks phase B and BSHOT clocks phase A. This technique of digital filtering insures repetitive edges on a single phase without an edge on the other phase are not passed on to the processor. Repetitive edges occur when the motor changes direction. Further logic obtains a direction or UP/DN bit. Note the first edge after a direction change is lost. A lost edge does not affect the count since the first transition is lost in both directions. Since an edge is lost in each direction, the circuit has an absolute resolution of one edge.

## 2.2 Interfacing to TIMER2

COUNT indicates an incremental position count on both its rising and falling edge. TIMER2 on the 80C196KB is a 16 bit externally clocked up/down counter clocked on the rising and falling edge of its input signal. A one or zero on port pin 2.6 determines whether TIMER2 counts up or down. By interfacing an optical encoder to TIMER2 as shown in Figure 8, an up/down counter is realized. No software intervention is required to keep track of position or direction changes with the 16 bit TIMER2. The CPU is free to concentrate on executing the control algorithm.

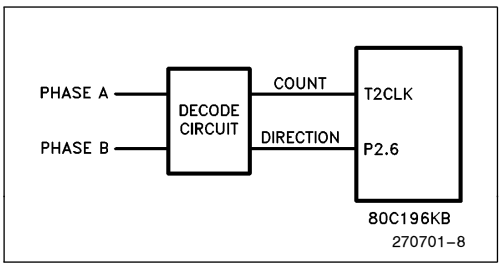



**Figure 8. TIMER2 and Encoder Interface Circuitry**

For designs requiring greater resolution, a 32-bit up/down counter may be realized with the same circuit and minimal software overhead. TIMER2 can cause an interrupt on an overflow condition. However, an overflow interrupt is not the safest way to implement a 32-bit up/down counter. Repetitive overflow interrupts could happen when the motor oscillates about a position where the LSW (Least Significant Word) is zero, or TIMER2 keeps overflowing and underflowing. For this method, the total software overhead required for a 32-bit up/down counter is dependent on the position and set point of the motor and would be difficult to predict.

A much better way to implement a 32-bit up/down counter is shown in Figure 9. TIMER2 is only read at the beginning of the control algorithm, or once a sample time. This does not present an accuracy problem for a digital control algorithm. TIMER2 is read into a temporary register. The temporary value is then subtracted from TIMER2, rather than clearing TIMER2, ensuring no counts will be missed. The 16-bit temporary value is sign extended to form a two's complement 32-bit value and added to the old 32-bit position value to form the current position value. This 32-bit up/down counter provides the accuracy needed for a control loop while keeping software overhead constant under all conditions.

A Pittman motor with a Hewlett Packard HEDS - 5310 512 line incremental shaft encoder was interfaced to TIMER2. Even at a maximum shaft rotation of 6000

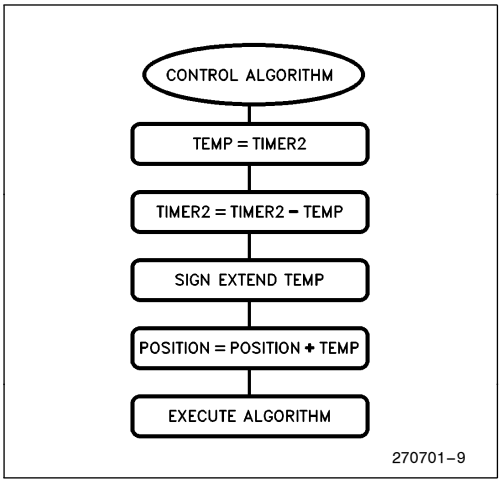



**Figure 9. Control Algorithm for TIMER2**

RPM, the edges are only clocked into TIMER2 at a period of about 5$\mu$s.

(6000 R/M) * (1/60 M/SEC) * (512 LINE) * (4 EDGES/LINE) = 204,800 edges per second

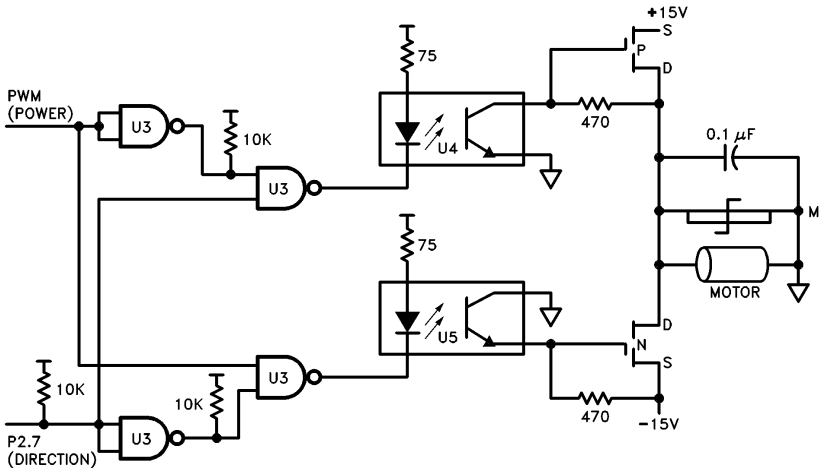
TIMER2 has a minimum transition period of once a state time, or 167 ns @ 12 Mhz, in the fast increment mode. Obviously, much higher resolutions and speeds may be obtained.

## 2.3 Interfacing to the HSI

The HSI can interface more than one motor to the 80C196KB. COUNT is input into an HSI pin which is configured to recognize events on both the rising and falling edge of its input signal. UP/DN is input to a port pin to determine direction. Up to four motors can be interfaced to the 80C196KB using the four input pins of the HSI. The disadvantage of using the HSI is an ISR (Interrupt Service Routine) must be executed on each edge. Considerable software overhead could occur if edges are clocked into the HSI faster than about one every 150$\mu$s.

Two Pittman motors with 2 line encoders were interfaced to the HSI to generate two 32-bit up/down counters as an example. With both motors turning at a maximum velocity of 6000 RPM, an edge will occur every 625$\mu$s. The ISR in Figure 10 processes the edges from the encoders and updates the position values and executes in about 15$\mu$s @ 12 MHZ on the 80C196KB. This still allows 97.6% (1 − 15/1250) of the total processing time to implement control algorithms and other I/O functions.

```
;;;
```




U3 = 7438
U4 = U5 = OC1H11A1
N = IR533 HEXFET
P = IR9533 HEXFET
M = Z47A7 MOV

**Figure 11. Motor Drive Circuitry**

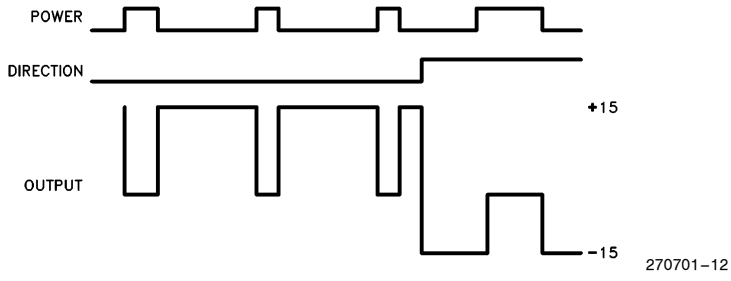



**Figure 12. Motor Drive Waveforms**

| Duty Cycle | PWM Register |
|---|---|
| 0% | 00 |
| 25% | 64 |
| 50% | 128 |
| 90% | 230 |
| 99.6% | 255 |

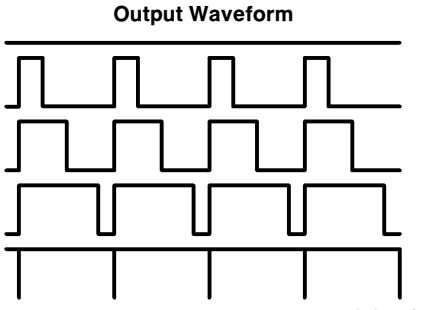



**Figure 13. PWM Output Waveforms**

The PWM unit along with pin 2.7 was used to drive one motor at a fixed output frequency of 23.6 Khz. By driving the motor at this frequency, motor whine in the audible range was eliminated. Note that a 00 value in the PWM register applies full power to the motor; the desired 8 bit output value must be inverted before it is loaded into the PWM__CONTROL register to obtain the correct output.

## 2.6 Using the HSO to Generate PWMs

The HSO (High Speed Outputs) of the 80C196KB can generate up to four PWMs. The HSO triggers events at specified times based on TIMER1 or TIMER2. For the specific purpose of generating PWMs, the event is driving an output pin high or low. HSO commands are loaded onto the CAM, (Content Addressable Memory), which specify the time and event to take place. The CAM is eight positions deep. The HSO triggers the event on a successful compare with the associated timer.

The 80C196KB can optionally lock commands onto the CAM. This feature is very useful for generating PWMs using TIMER2 as the time base. Figure 14 shows an example of two PWM outputs using locked commands in the CAM. TIMER2 is clocked externally at a frequency which determines the resolution of the PWMs. TIMER2 can be clocked at a maximum frequency of once every eight state times (1.33$\mu$s @ 12 Mhz) when used with the HSO. The RESET TIMER2 @ T4 command specifies the output frequency of the PWMs. By changing the external TIMER2 clock frequency and the value of T4, the HSO can generate a wide range of PWMs.

T0 and T1 specify when the output pins will be driven low. By varying T0 and T1, the duty cycle of the output waveforms are changed. Both pins are driven high by the same command at the same time TIMER2 is reset. Since there are still four positions open in the CAM, two more PWMs could be generated and one position would still be left open in the CAM.

For this ap-note, two Pittman motors were controlled using the HSO along with port pins 2.6 and 2.7. It was desired to keep the output frequency the same as the output frequency of the on-board PWM. To accomplish this, TIMER2 was clocked every 8 state times and reset when it reached 31 counts. This makes the output frequency 23.6 Khz @ 12 Mhz with 5 bits of resolution. CLKOUT was externally divided by 16 and input into TIMER2. Since TIMER2 counts on both the positive and negative edge of its input signal, a square wave with a 16 state period clocks TIMER2 every 8 state times.

The ISR used to load commands onto the CAM is shown in Figure 15. When the control algorithm determines an output has changed, a RESET TIMER2 command gets loaded onto the CAM to generate an interrupt. The interrupt vectors to this routine and updates the CAM. To clear a locked entry from the CAM, the entire CAM must be flushed by setting IOC2.7. Care must be taken to reload all of the commands. This includes any commands not locked on the CAM.

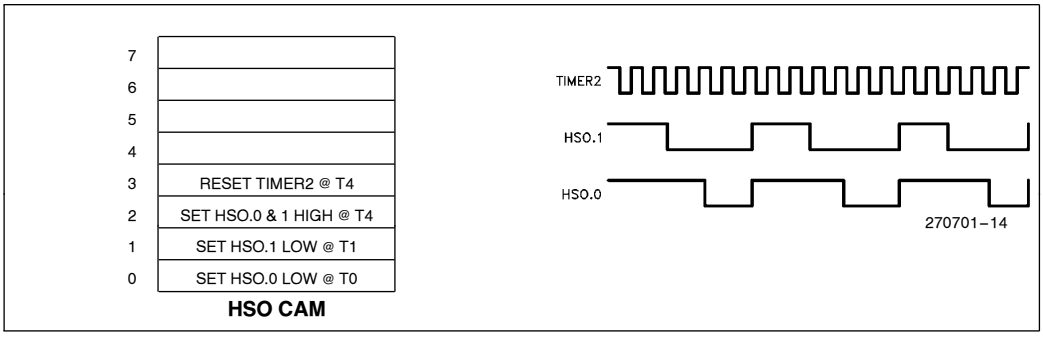



**Figure 14. Two PWMs Using HSO Locked Entries**

```
timer2_reset:
        ldb   IOC2,#11000000b                  ;clear the CAM
        ld    hso_command,#11001110b           ;load reset timer2 command
        ld    hso_time,#31
        nop
        nop
        ldb   hso_command,#11100110b           ;this command will set both
        ld    hso_time,#31                     ;hso lines for the PWM

                                               ;load mot_4_power value
        cmpb  mot_4_power,#31                  ;if power is 1fh, do not load
        je    check_4                          ;this command, it will cancel
       ldb   hso_command,#11000000b            ;with the set command
        ldbze hso_time,mot_4_power
check_4:
                                               ;load mot_5_power value
        cmpb  mot_5_power,#31                  ;if power is 1fh, do not load
        je    sanity_check                     ;this command, it will cancel
        ldb   hso_command,#11000001b           ;with set command
        ldbze hso_time,mot_4_power

sanity_check:
        cmp   TIMER2,#32                       ;sanity check to make sure
        jnh   sane                             ;TIMER2 is not greater than 3
        clr   TIMER2
sane:
        ldb   hso_command,#39h                 ;reload software timer 1
        add   swt1_period_bak,#swt1_dly_period
        ld    hso_time,swt1_period_bak
        ldb   port2,port2_bak                  ;load direction bits
        popf
        ret
```

270701-15

**Figure 15. HSO Interrupt Service Routine**

There is the potential for commands to be missed when they are flushed and reloaded on the CAM. For example, an HSO command is loaded on the CAM to clear HSO pin 3 when TIMER2 = 23 and the CAM is flushed when TIMER2 = 22. A new HSO command is then loaded onto the CAM to clear HSO.3 when TIMER2 = 21. This command will not execute until TIMER2 is cleared and counts back up to 21. Missed commands are difficult to avoid without excessive software overhead. Software must take missed commands into account and minimize the effects on the application.

The ISR in Figure 15, insures if an output edge is missed for one period of TIMER2, the HSO pin will remain high. A logical one applies no power to the motor. Also, at the end of the routine a sanity check makes sure TIMER2 is not greater than 31.

## 2.7 Current Limiting

When a motor is stalled, or excessively loaded, it will draw a lot of current. Current limiting can be used to keep the motor from damaging itself, or anything in its path. Several options exist to the user on what to do about a high current condition. Less power could be applied, or the motor could shut off entirely. This section only explains how to recognize a high current condition in a DC servo motor, not what to do about it.

Figure 16 shows a way to convert the current from the motor into a voltage which can be read by the 80C196KB onboard A/D converter. Again, an opto-isolator keeps the motor and digital power supplies separate. When enough current flows through the opto-isolator, the A/D input voltage will drop down to about .7 volts. The current to the opto-isolator is varied by changing the values of the two resistors, R1 and R2 which split the current flow. By changing R1 and R2, this circuit can be adjusted to work properly with different motors and load conditions.

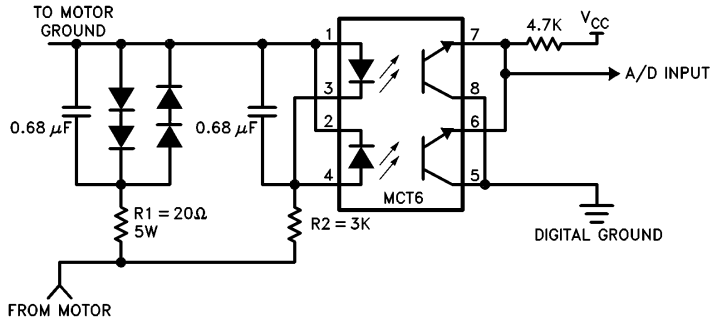



**Figure 16. Current Sensing Circuitry**

Motor startup current must be considered when testing for a high current condition. When a motor is started, it will draw a great deal of current. This current surge can last for a few milliseconds. Software must decide if the motor is drawing excessive current because it is stalled, or just starting. The section of code in Figure 17 executes during the control algorithm. The current must be above ad__limit for 30 sample times before software recognizes a high current condition. Of course, these values must be adjusted up or down depending on the motor and load conditions.

```
;do a current limit check

       jbs   ad_command,3,motor_around   ;if A/D still running,skip
       cmpb  ad_result_hi,ad_limit
       jh    current_ok
       incb  ad_count                    ;want to do 30h A/D conversions
       cmpb  ad_count,#30                ;before acting because of motor
                                         ;startup current

       jne   current_maybe_ok

;here is where the user inserts his code on what to do
;about a high current condition

current_ok:
       clrb  ad_count
current_maybe_ok:
      ldb    ad_command,#00001001b       ;start another a/d conversion
motor_around:
```

270701–17

**Figure 17. Current Sensing Software**

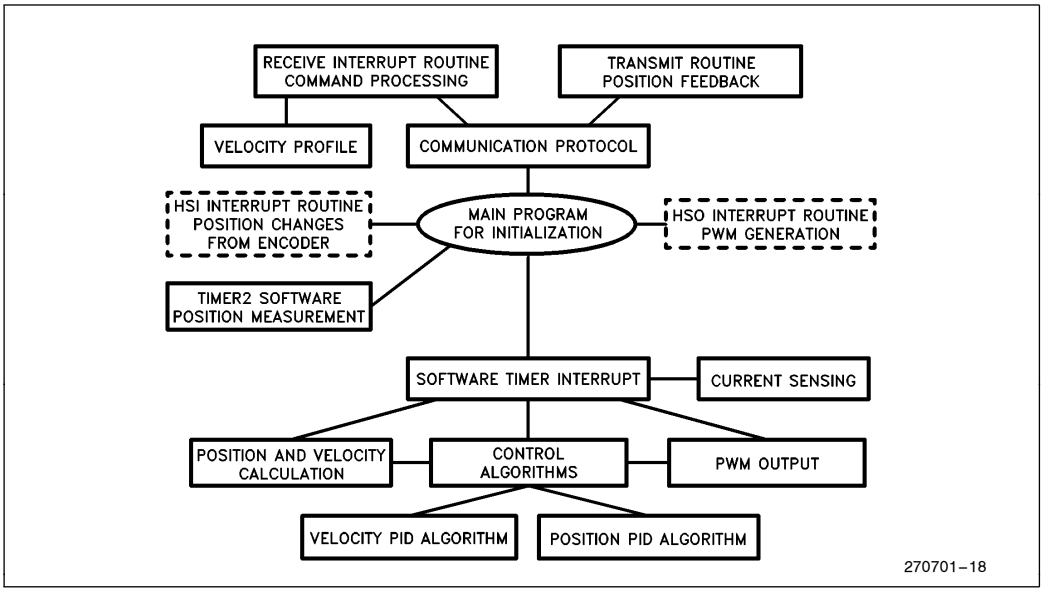



**Figure 18. Software Block Diagram**

## 3.0 SOFTWARE

A block diagram of the software is shown in Figure 18. The software consists of a main program for hardware and software initialization of the 80C196KB peripherals and programming of control tasks. The control tasks include tracking the motor position and direction, control of the motor speed and direction, detection of overrun conditions and communication to the master controller. After initialization is complete, the 80C196KB enters idle mode to preserve power while not performing control tasks. Interrupt service routines for the serial port, HSI, HSO and software timer perform the various control tasks.

The communication protocol to the main controller is implemented in the serial receive and transmit routines. Commands from the master controller move the motor in one of two modes, manual or automatic, depending on the command. The commands are listed in Figure 28.

Manual mode moves the motor clockwise or counterclockwise with a preset maximum control voltage applied. Manual mode commands include MOTOR UP, MOTOR DOWN and STOP. The MOTOR UP and MOTOR DOWN commands send the motor into manual mode. The motor continues to run in the desired direction until a STOP command is issued from the master controller. The STOP command loads the destination position with the current position and enters automatic mode.

Automatic mode positions the motor using either a position or velocity PID algorithm. The position PID algorithm is applied after reception of the STOP command or when the desired position is reached. The destination position can be changed by a POSITION command from the master controller.

The maximum motor velocity and the destination position are contained in the POSITION command. If the maximum velocity is zero, a position PID is applied to move the motor to the destination position. A non zero maximum velocity will position the motor using a velocity PID algorithm. Position and velocity input to the algorithms are calculated based on position input from the encoder.

Position information for the PID algorithms can be provided by using the High Speed Inputs or TIMER2. The HSI interrupt routine processes the direction and position information incoming from the encoder to provide current motor position. Alternatively, TIMER2 directly measures the position when used as an up/down counter. Velocity information can be calculated using the position information given a constant sampling rate. The position and velocity information are used by the PID control algorithms.

The control algorithm uses a software timer interrupt to generate the sampling rate of the control software. The main portion of the software timer routine calculates the current position and velocity, senses the motor

current for overrun conditions, calls the PID control algorithm and generates the PWM control voltage to the motor.

The speed of the motor can be controlled using the PWM or the HSO. If the HSO is used, the HSO interrupt routine generates a PWM output to control the voltage applied to the motor. Otherwise, the PWM unit controls the voltage applied to the motor.

Each of the major software routines is covered in detail in this section.

## 3.1 Main Initialization Routine

The main initialization routine executes immediately following reset to initialize the 80C196KB peripherals and enable the interrupt driven control tasks. A flow chart for the main initialization routine is shown in Figure 19. The constants and variables for the control algorithms and software routines are loaded into register space for fast execution.

Next, the various peripherals are programmed to handle the control tasks. The PWM for voltage control of the servo motor is initialized. TIMER2 is programmed as an up/down counter with T2CLK as the clock source. The serial port is set to 19.2 Kbaud and programmed for mode 2 to receive incoming commands. An A/D conversion is started to check for initial stress conditions. Before the motor can be accurately positioned, an initial reference point must be established.

In order to find the reference point, an I/O port is connected to a limit switch. The motor is driven in a preset direction until the limit switch is activated. The initial position is then loaded and position PID control is applied to keep the motor stable. Position commands from the master controller can now precisely position the motor from the established reference point.

Finally, the software timer, timer overflow, receive and transmit interrupt routines are enabled and the idle mode is entered to conserve power. The routines will execute as each individual interrupt control task requires servicing. Discussion of the control tasks of each software routine is contained in the following sections.

## 3.2 Software Timer Interrupt Routine

The software timer interrupt service routine executes every 500 $\mu$s and determines the sampling rate of the PID control algorithm. Figure 20 shows the flow chart for the software timer interrupt routine. The routine determines the operating mode, calculates the current velocity and position and tests for overrun of preset boundary conditions and stress conditions.

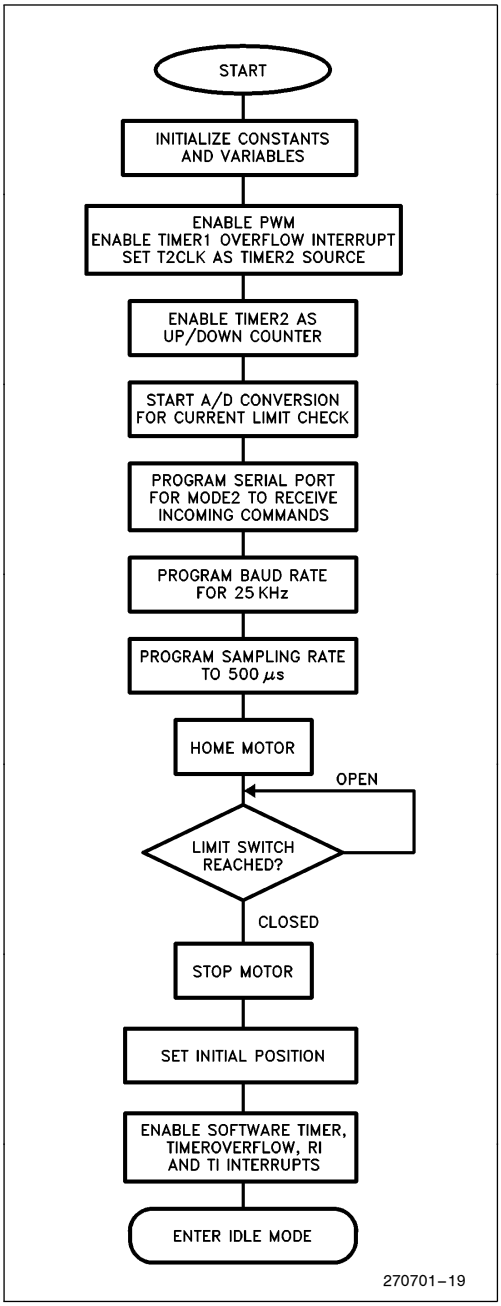



**Figure 19. Motor Initialization Routine**

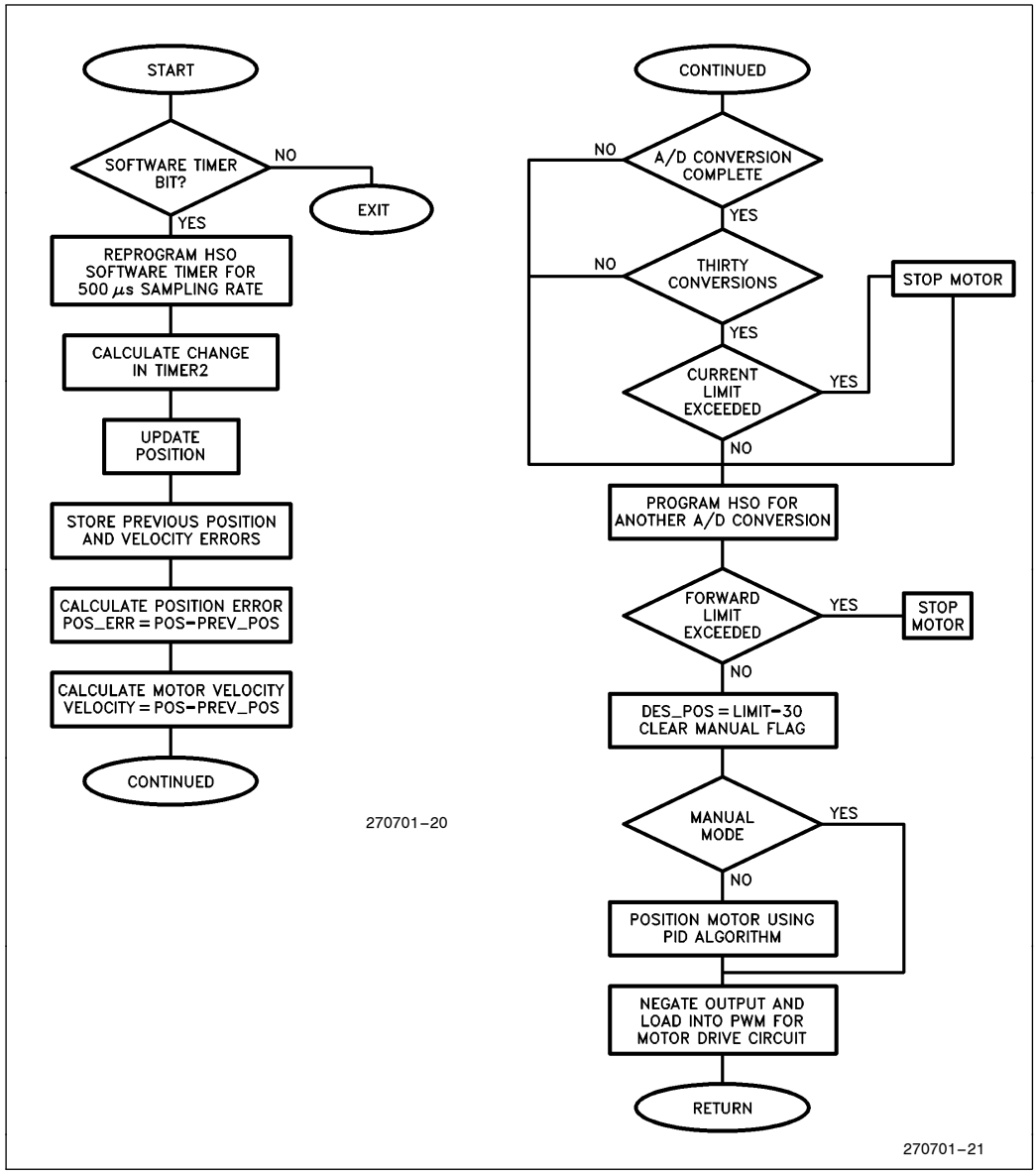



**Figure 20. Software Timer Interrupt Routine**

An A/D conversion compares the motor current to test for a stress condition against a preset limit. Thirty conversions are done to average the motor current to prevent a false trigger due to a large current surge when the motor starts up. If the preset limit indicating a stress condition is exceeded, the motor is stopped.

The motor is also stopped if the current position exceeds the preset boundary limits. In the case of the robot, the movement of joints are limited to prevent positions which may cause stress conditions or damage the robot. The positioning of the robot is dependent on the mode of operation.

A manual flag is tested to determine if the automatic or manual mode should position the motor. The manual mode moves the motor either up or down with a preset maximum motor control voltage until a STOP command is issued. The automatic mode positions the motor using either the position PID for accurate positioning or the velocity PID for long positioning.

The software timer interrupt routine calculates and stores the current position and velocity of the motor for use by the appropriate PID algorithm. The current velocity is calculated given the sampling rate, the current position and the previous position. The calculated velocity and position information is stored in the 80C196KB register space for use by the PID algorithm software.

Recall that either a position PID or a velocity PID control algorithm will be executed depending on the maximum velocity value passed by the master controller. If the value is zero, a position PID is employed, otherwise, the velocity profile is employed. The velocity profile PID is ideal for large maneuvers while the position PID is better for shorter movements or maintaining the current position. The generated output from the control algorithm is then loaded into the PWM control register and a return is executed.

## 3.3 PID Control Algorithm

The algorithm used to control the angular position and velocity of the motor is a common PID algorithm. The algorithm uses proportional, integral and differential feedback to control the output to a motor. The PID algorithm controls the important system characteristics of the motor: settling time, steady state error, and system stability. Each term in the control algorithm affects each system characteristic differently. A block diagram of the PID algorithm is illustrated in Figure 21.

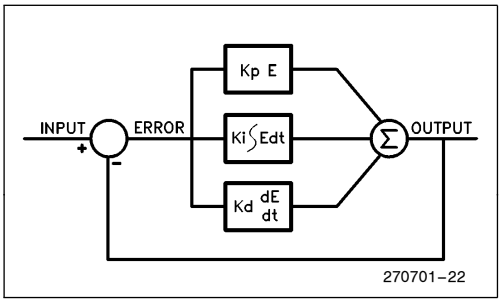



**Figure 21. Block Diagram of PID Algorithm**

The PID algorithm consists of three terms: a proportional term, integral term and differential term. The proportional term drives the motor with an output directly proportional to the error between the desired and measured position. The integral term consists of the integral of the position errors multiplied by an integral constant. The differential term is the change in error multiplied by a differential constant. The sum of the terms is then scaled to provide a control voltage to the motor. The system characteristics of the motor are tuned by the selection of appropriate constants.

The settling time, steady state error and system stability are impacted by the amount of proportional gain selected. To accurately control a small change in motor position, a large gain is desired. Faster system response is attained by selecting a large gain but at the cost of greater overshoot and longer settling time. The effect of varying loads on the motor makes proportional control in itself inadequate because of system instability and large steady state error.

Application of integral feedback drives the steady state error to zero by increasing the output in response to a steady state error. The integral term increases as the sum of the steady state error increases causing the error to eventually be driven to zero. The integral term, although driving the steady state error to zero, can cause overshoot and ringing if it is too large. This has the undesirable affect of poorer system response. Applying PI control works very well, however a faster system response can be acheived by applying a PID algorithm.

System response can be improved by adding a differential term. Addition of this term improves the response time by providing a output proportional to the rate of change in error. When the motor has a large change in error, the term produces a large output to the motor. Therefore, the system responds faster to disturbances in the system. Most of the system instability is caused by too high of a differential constant. The size of the proportional, integral and differential constants provide tradeoffs to the desired system characteristics.

Selection of the three gain constants is critical in providing fast system response with good system characteristics. A slightly modified PID algorithm controls the motor which improves both the system response and the system stability. Two modifications were made to improve the control algorithm. First, the size of the integral term was clamped to prevent instability caused by an extremely large integral term which could occur after a long time with large errors. Second, the integral term was cleared when the error changed sign to further improve the system stability. The PID control algorithm is written in PL/M-96 for ease of development.

## 3.4 Position PID Software

The software flow chart for the PID algorithm is shown in Figure 22. Upon entering the routine, the position

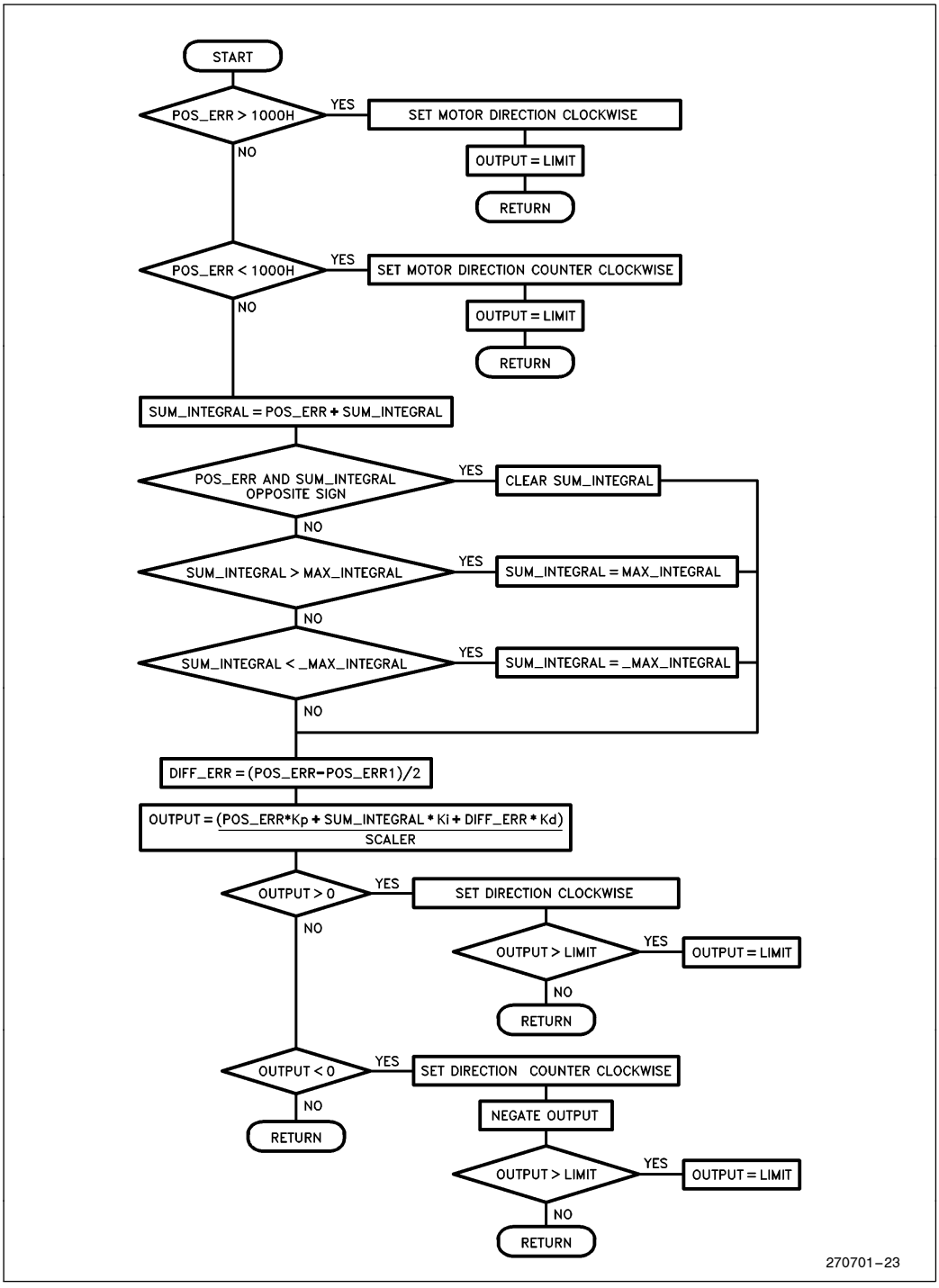



**Figure 22. Position PID Algorithm**

error is checked for a minimum value before applying the position PID algorithm. If the minimum position error is exceeded, the maximum PWM output is applied to move the motor as rapidly as possible.

Current position error is added to the integral sum. Position error and integral sum are tested to clear the integral sum if they are opposite in sign. This improves the system stability by preventing the integral term from applying a correction opposite to the desired output.

If the integral sum is greater than the maximum sum allowed, the integral sum is clamped. This prevents the integral sum from becoming too large if the error is large for several samples. Differential error is then calculated from the current and previous position errors.

Output for the PID algorithm is calculated from the proportional, integral and differential terms multiplied by their individual gain constants. The output is then scaled and tested for the preset PWM output limit. If the limit is exceeded, the output to the PWM is set to the maximum value. The appropriate motor direction is set depending on the sign of the output. The final output to the PWM control is ready and the software returns.

## 3.5 Velocity Profile

Positioning of a servo motor using only a position PID algorithm wastes power and gives poor system performance when moving between two positions. A velocity profile provides a smooth transition between two angular positions and improves the energy consumption of the motor. Three different velocity profiles which can be applied are trapezoidal, triangular and parabolic.

The parabolic profile is the most power efficient and provides smooth acceleration and deceleration at the end points. However, a large amount of processor time is needed to calculate the profile in real time. The triangular profile provides ease of calculation versus the parabolic but generates a rough transition at the peak of the profile. A trapezoidal profile provides energy efficiency, ease of calculation and relatively smooth acceleration and deceleration throughout the velocity profile. For these reasons, the trapezoidal profile was selected.

A trapezoidal profile consists of an acceleration period, run period and deceleration period. The variables ACCEL__TIME, RUN__TIME and END__TIME represent the periods. Figure 23 shows the trapezoidal profile. Acceleration and deceleration rates for the motor are fixed according to the optimum values found through testing. The master controller sends a position command containing the maximum velocity (MAX__VELOCITY) and the desired end position (DES__POSITION). The DES__POSITION is equal to the integral of the velocity profile (i.e., the final position can be determined by integrating the velocity over the period of the profile. Therefore, the ACCEL__TIME, RUN__TIME and END__TIME can be calculated based on the DES__POSITION, ACCELERATION, DECELERATION and MAX__VELOCITY.

The destination position should be reached if the velocity profile was ideally tracked. However, a certain amount of position error can be expected as the motor travels from one point to another. This error is eliminated by applying the position PID at the end of the velocity profile. This modified control algorithm has both good motor performance and accurate angular positioning.

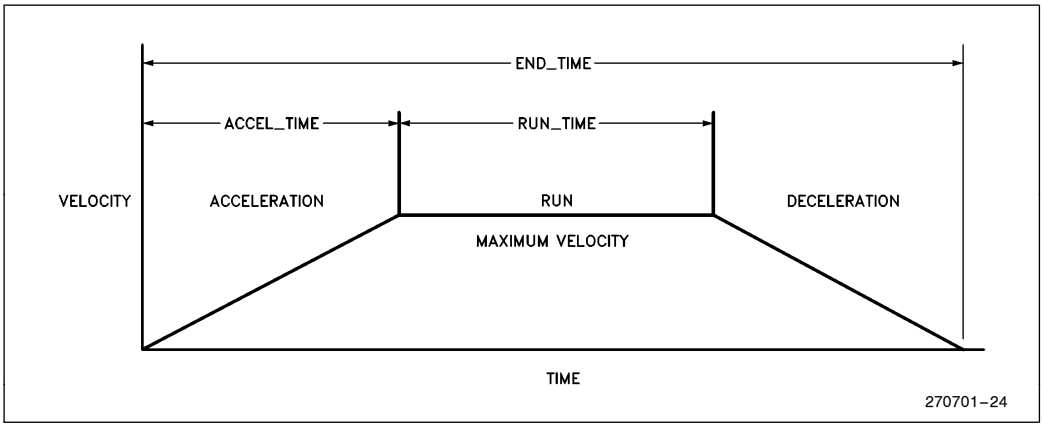



**Figure 23. Trapezoidal Velocity Profile**

## 3.6 Trapezoidal Profile Calculation

The trapezoidal velocity profile is calculated when a position command with a nonzero maximum velocity is passed from the master controller. The master passes the desired end position and the maximum velocity of the motor. A reasonable acceleration (deceleration) rate was found through experimentation to be 1 position count/sampling rate (500 $\mu$s). ACCEL__TIME, RUN__TIME and END__TIME can be easily calculated given the relative acceleration rate of one, the end position and the maximum velocity.

The acceleration and deceleration time is equal to the maximum velocity since the acceleration/deceleration rate is one. RUN__TIME is the difference between the desired position and current position minus the distance covered during the acceleration and deceleration times. END__TIME is the RUN__TIME added to two times the ACCEL__TIME. With the velocity profile calculated, the velocity PID algorithm will be applied until the END__TIME is reached.

The velocity profile software generates the appropriate velocity depending on the current time. Figure 24 shows the velocity profile generation software. The TIME variable is incremented every software timer interrupt at the sampling rate if it is less then the end time (END__TIME) of the profile. Three different velocities are calculated during the profile. DES__VELOCITY equals the ACCELERATION multiplied by the TIME until the ACCEL__TIME is reached. The DES__VELOCITY equals the maxiumum velocity until the RUN__TIME is exceeded. Once the RUN__TIME is exceeded, the velocity is equal to the ACCELERATION (same as deceleration rate) multiplied by the TIME-CURR__TIME. When the end of the profile is reached (which is approximately the desired end position), the time equals the END__TIME and the position PID controls the motor. If the maximum velocity passed by the master controller is zero, the CURRENT__TIME is set to the END__TIME and the position PID controls the motor.

The velocity control algorithm employs the PID algorithm. The algorithm is similar to the position algorithm used to control the position. The velocity control algorithm is shown in Figure 25.

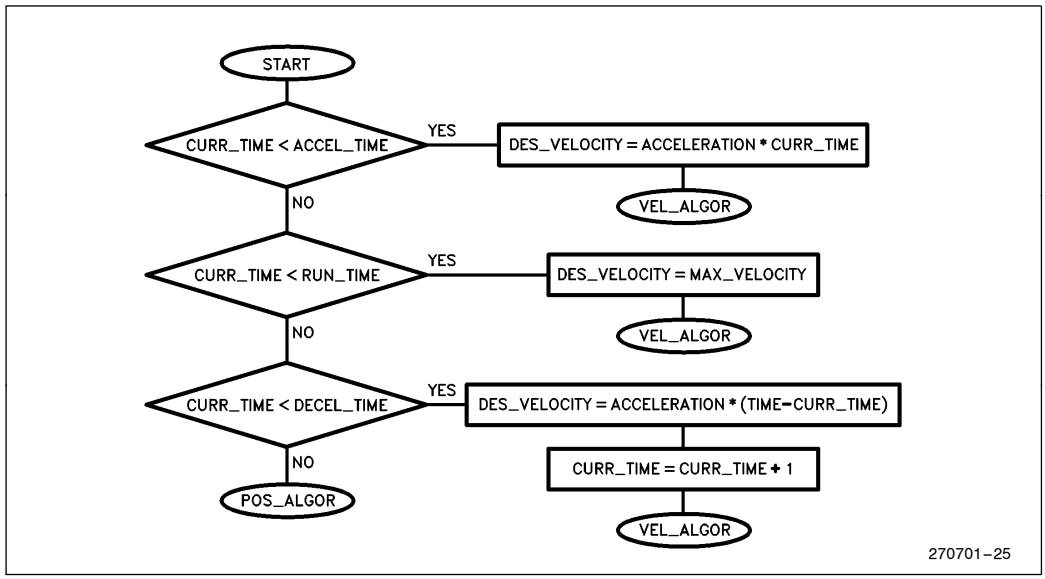



**Figure 24. Velocity Profile Generation Software**

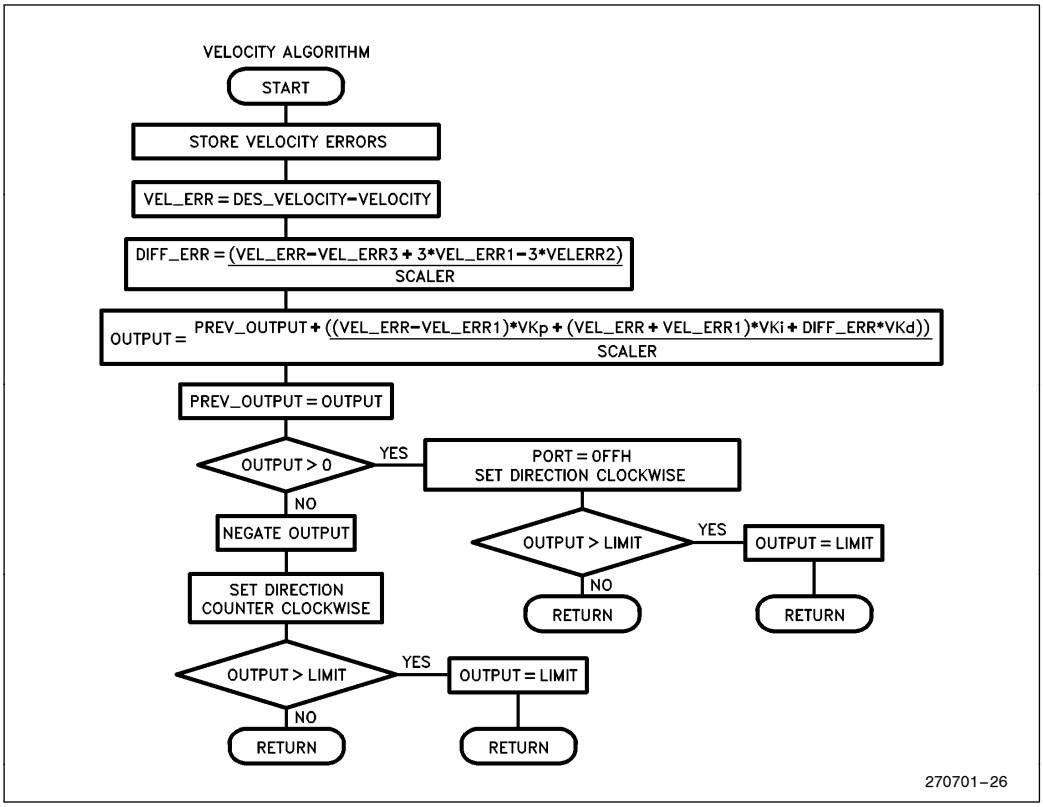



**Figure 25. Velocity Control Algorithm**

## 3.7 Fast Execution of Control Algorithms

The high speed arithmetic operations capability, availability of three operand instructions and large register space of the 80C196KB provide for fast execution of control algorithms. The 80C196KB running at 12 Mhz can execute a 16 × 16 Multiply in 2.3 μs and 32/16 divide in 4.0 μs. Three operand instructions operate on two variables without modification and store the result in the third variable. This eliminates the need for executing load and store operations as required by accumulator bound architectures. The large register space can store all of the constants and variables for the control algorithm without the use of load and store operations. In addition, procedures do not need to pass parameters or store results since they can permanently reside in register space.

A summary of the execution times for the main software routines is shown in Figure 26.

| | Execution Time |
|---|---|
| Software Timer Interrupt Routine | 40 μs |
| PID Control Algorithms:<br>Velocity PID (PL/M-96/ASM-96)<br>Position PID (PL/M-96/ASM-96) | <br>300 μs/30 μs<br>240 μs/40 μs |
| Velocity Profile Generation | 71 μs |
| HSI Interrupt Processing | 22 μs |
| HSO Generate PWM Routine | 16 μs |
| Receive Interrupt and Command Processing | 26 μs |
| Transmit Interrupt Routine | 11 μs |

**Figure 26. Execution Times for Main Software Routines**

The HSI, HSO, Receive and Transmit Interrupt routines take a minimal amount of time. A majority of the processing time is in executing the Software Timer interrupt routine and either the Velocity PID or Position PID control algorithms.

PID Control Algorithms take a considerable amount of time since they are written in a high level language and execute a number of thirty-two bit arithmetic operations. Thirty-two bit accuracy is not required since the maximum position required to accurately track the motor is about twenty four bits. To optimize the control algorithm for the accuracy required, the routines can be written in assembly. A sample Position PID algorithm is shown in Figure 27. The routine executes in about 30 $\mu$s by optimizing the control algorithm and minimizing the number of 32-bit operations.

```
VPID:       ld vel_err3,vel_err2                ; store velocity errors
            ld vel_err2,vel_errl
            ld vel_errl,vel_err
            sub vel_err,des_velocity,velocity   ; calculate velocity error

            sub temp,vel_errl,vel_err2          ; calculate differential error term
            mul temp,#3       ; diff_err=(vel_err-vel_err3+3*vel_errl-3*vel_err2)
            sub temp,vel_err3
            add temp,vel_err

; Output=prev_output + ((vel_err-vel_errl)*VKp+(Vel_err+Vel_errl)*Vki + diff_err*Vkd))/
;scaler

OUTPUT:     mul temp,Vkd                        ; calculate differential term
            add temp2,vel_err,vel_errl
            mul temp2,Vki                       ; calculate integral term
            add temp,temp2
            sub temp2,vel_err,vel_errl          ; calculate proportional term
            mul temp2,Vkp
            add temp,temp2
            div temp,scaler                     ; scale output
            add output,prev_output,temp
            ld prev_output,output
            div 0ut_scaler                      ; Scale 32 bit output to get l6 bit result
            jbc 0ut+3,7,forward                 ; test output for direction
REVERSE:    neg 0ut+2                           ; negate output
            ldb p2,#07fh                        ; set direction down(p2.0=0)
            sjmp scaleout
FORWARD:    ldb p2,#0FFh                        ; set direction up(P2.0=l)

SCALE0UT:   cmp 0ut,#0ffh                       ; scale output for maximum pwm value
            jgt exit                            ; if 0ut > maximum pwm output
            ld 0ut,#0ffh                        ; then clamp output to max pwm value
EXIT:       ldb pwm,0ut
            ret
```

**Figure 27. Position and Velocity PID Assembly Routine**

```
PID:      add sum_int, pos_err          ; sum position errors
          div sum_int,decay             ; limit effect of old position errors
          sub diff_err, pos_err         ; differential error = (pos_err - pos_errl)/2
          div diff_err, #2
;         Out = Kp*pos_err + Ki*interr + Kd*differr
OUTPUT:   mul Out pos_err, Kp           ; Calculate proportional term
          mul temp, Ki interr           ; Calculate integral term
          add Out, temp                 ; add integral term to Output
          addc Out+2, temp+2            ; 32 bit add to maintain full 32 bit accuracy
          div Out, scaler               ; Scale output
          jbc Out+3,7,forward           ; test output for direction
REVERSE:  neg Out+2                     ; negate output
          ldb p2,#07fh                  ; set direction down (P2.7=0)
          sjmp scaleout
FORWARD:  ldb Port2,#0ffh               ; set direction up(P2.7=1)
SCALEOUT: cmp Out,#0ffh                 ; scale output for maximum pwm value
          jgtexit                       ; if Out > maximum pwm output
          ld Out,#0ffh                  ; then limit output to maximum value
EXIT:     ldb pwm, Out                  ; load pwm with Output value
          ret
```

**Figure 27. Position and Velocity PID Assembly Routine** (Continued)

## 4.0 Distributed Control

Distributed control of servo motors requires the passing of commands and data from a master to a slave. The master passes commands to report position, start and stop the motor, or position the motor to an exact location using a position PID or velocity profile. The slave needs to report current position and acknowledge incoming commands from the master. This protocol requires addressing of slaves and the distinction between incoming commands and transmission of data. The 80C196KB serial port provides a multiprocessor communication mode for implementing the protocol.

The 80C196KB provides a ninth bit in Mode 2 and Mode 3 that can assist communication between multiple processors. If the received ninth bit is zero in mode 2, the serial port interrupt will not occur. Each motor is initially programmed for this mode to distinguish receiving a command versus a data byte. With the ninth bit set, indicating a command byte has been received, all the slaves interrupt and process the incoming byte. The address of the motor being controlled is embedded in the command byte. All processors will process the command byte if the motor address matches.

A motor receiving a poll command from the master controller will enter mode 3. The polled motor then receives the data bytes which are sent with the ninth bit cleared. Therefore, only the processor receiving data will interrupt for serial reception while the other processors await another command byte with the ninth bit set. A list of available commands and the format for each is shown in Figure 28.

**Command Table**

| Command | Code | Operation |
|---|---|---|
| Position | 01 | Position motor using either position PID or Velocity profile. |
| Poll | 05 | Polls motor for current position. |
| Motor Up | 08 | Enters manual mode turning motor clockwise. |
| Motor Down | 09 | Enters manual mode turning motor counter clockwise. |
| Stop | 10 | Exits manual mode setting the desired position to the current position. |

**Position Command**

| Command | Position | Maximum Velocity |
|---|---|---|
| 01 | 4 bytes | 2 bytes |

**Poll Command**

| Command | Position |
|---|---|
| 05 | 4 bytes |

**Figure 28. Master Commands and Format**

## 4.1 Receive Interrupt Service Routine

Communication between the 80C196KB and the main controller is handled by the serial port routine. Figure 29 shows the flow for the receive interrupt service routine. Upon reception of a byte from the main controller, a receive interrupt will occur. The RI bit is tested to ensure a byte has been received. If a byte has not been received, an error is generated and a return from the routine is executed. After a valid reception, the ninth bit is tested to determine if the incoming byte is a command byte or incoming data sent after reception of a POSITION command.

If the byte is a command byte, the motor address is checked by each slave for its own address. The command byte is then echoed back to the master controller by the appropriate slave. The routine is exited if the command byte is not for the motor. Since each motor has a unique address, only the motor receiving the command will respond. Reception of a POSITION command will switch the serial port to mode 3.

Desired position and maximum velocity is sent by the master to each slave by a POSITION command. Received data for the position command is stored in a buffer. After all data has been received, MAX_VELOCITY and DES_POSITION is loaded with the values stored in the buffer and the serial port is switched back to mode 2.

Each command is then checked and appropriate action taken depending on the received command. Commands include POSITION, POLL, UP MOTOR, MOTOR DOWN and STOP. The commands are summarized in Figure 28.

## 4.2 Manual Positioning

The receive routine will check for one of three manual commands: MOTOR UP, DOWN MOTOR or STOP. A manual flag is used by the software determine if the motor should be positioned using either a position or velocity PID algorithm or by manual control. The motor up and motor down commands set the manual flag which will cause the PWM control to be loaded with a constant value during the software interrupt routine. The direction port bit is set to the appropriate value depending on whether the command is up or down. The motor will continue to move up or down until a STOP command is issued by the master controller or the motor's preset limits are reached.

A stop command will reset the manual flag and set the controller in automatic mode which employs the PID algorithm. The destination position gets loaded with the current position and a return from the receive interrupt is executed. The manual position mode is used by the master controller to position the motor under keyboard or switch control. This is instead to precise position control of the motor by sending a position command.

## 4.3 Motor Positioning

Either position control or a velocity profile can be used to position each motor. The maximum velocity information stored in the POSITION command determines the type of method employed. If the maximum velocity value is nonzero, the velocity PID algorithm will be applied to position the motor. If the maximum velocity is zero, position control using the PID algorithm will be used. This provides for two alternative methods for positioning the motor.

Once a POSITION command is received, the processor enters serial mode 3 to receive the incoming position and maximum velocity information. The four bytes of position data and two bytes of maximum velocity are retrieved from a six byte storage buffer. A receive count keeps track of the number of incoming bytes until all bytes of the six byte frame have been received. If a frame or overrun error occurs, the motor will shut off and a 0FFH will be transmitted back to the master controller to indicate an error condition has occurred. Otherwise, an 88 is returned to indicate the valid transmission of position and maximum velocity. The manual flag will be turned off and the appropriate PID algorithm will be applied on the next software interrupt.

## 4.4 Master Polling of Position

The master controller can poll each motor controller for position with a poll command. After reception of the poll command, a transmit buffer is loaded with four bytes of position information. Each byte is then transmitted using the transmit interrupt routine.

The flowchart for the routine is shown in Figure 30. The routine simply tests the TI flag and continues to transmit a byte from the buffer until the transmit count goes to zero. After the count goes to zero, the transmission is complete and processing continues.

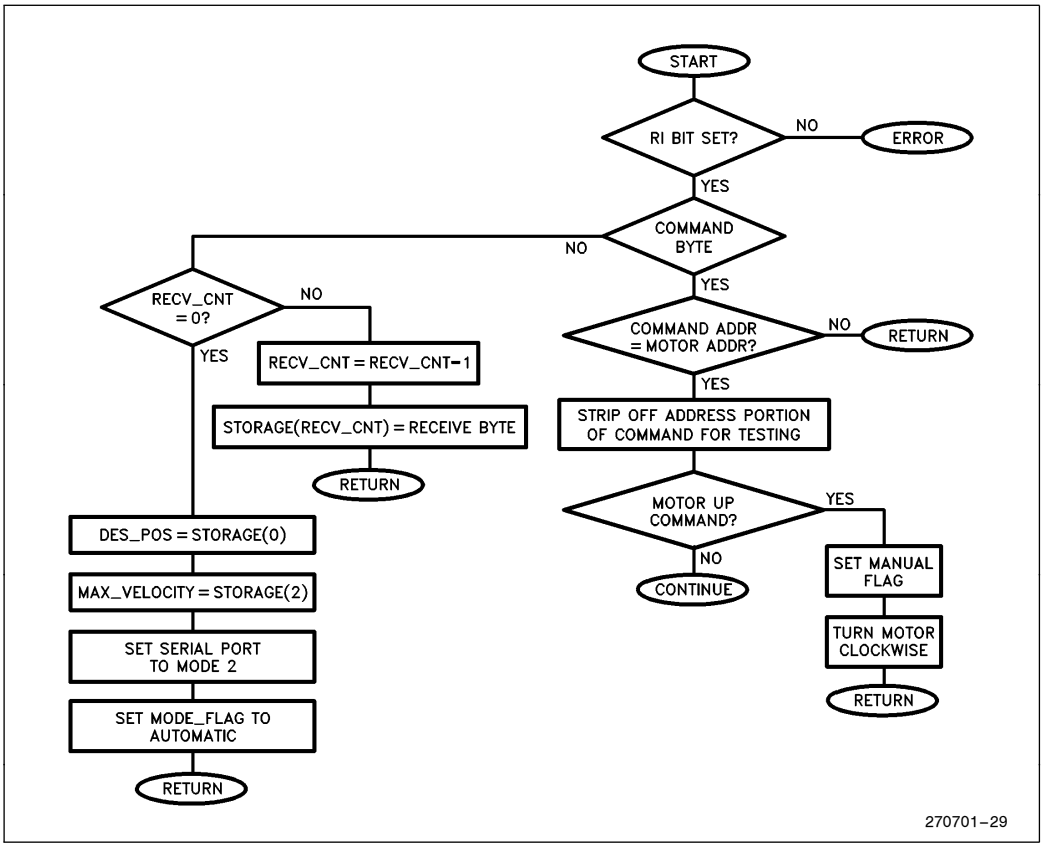



**Figure 29. Serial Port Receive Interrupt Routine**

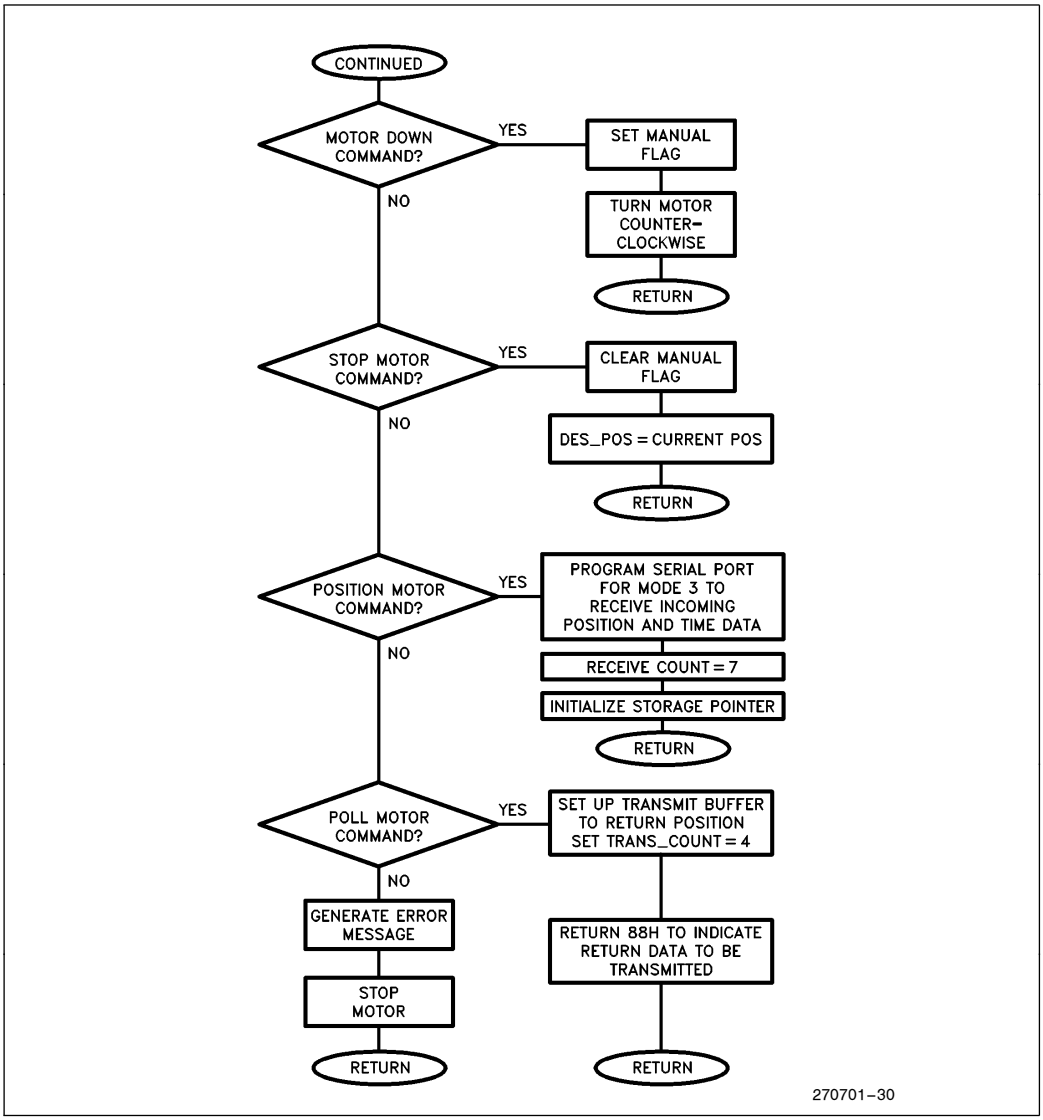



**Figure 29. Serial Port Receive Interrupt Routine** (Continued)

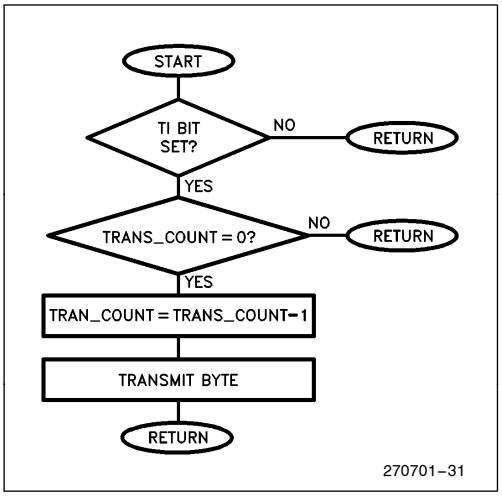



**Figure 30. Serial Transmit Routine**

## 5.0 DISTRIBUTED CONTROL OF A SIX AXIS ROBOT

A six axis robot demonstration system was built using distributed control of its six motors. The robot is a RHINO XR-1 prototype robot designed by SANDHU Machine Design Inc. Robot motors were replaced with similar models with high resolution encoders. The robot allows movement along six joints: base, shoulder, elbow, wrist, hand and fingers. Each joint is connected to a motor. The system used an IBM PC acting as a master controller.

The software used to develop the human interface was Turbo Prolog and the Turbo Prolog Toolbox. The human interface allowed for the programming and movement of the robot by individually controlling each joint motor. The IBM PC controlled each axis of the robot by passing commands serially.

The IBM PC provides a flexible master controller for positioning the robot. There are a large number of software languages for developing the control algorithms and human interface of the master controller. Turbo Prolog was selected for its low cost and ease of implementation. The control screen and robot programming language were rapidly developed using the Turbo Prolog. The software and hardware implementation easily provide for programming and controlling the robot through a variety of repetitive tasks. A robot using this control system could easily perform assembly or manufacturing tasks as shown in Figure 31.

### 5.1 Hardware Interface

The hardware interface to the robot is shown in Figure 32. Each major joint, elbow wrist, base and shoulder were controlled with a single 80C196KB using the PWM and TIMER2 as an up/down counter. The hand and finger motors used the HSI to track position and the HSO to generate PWM motor control voltages.

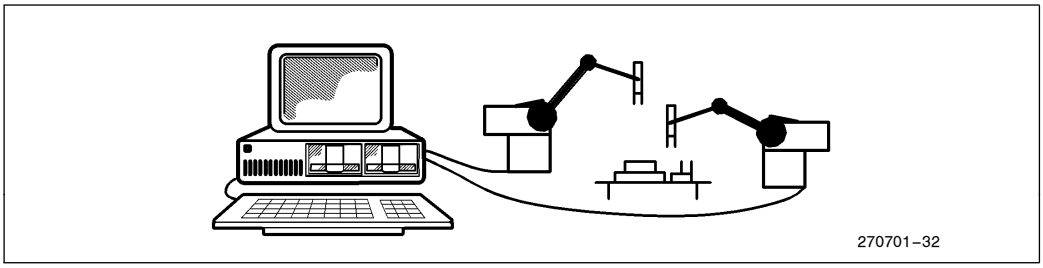



**Figure 31. Automated Assembly using Distributed Control**

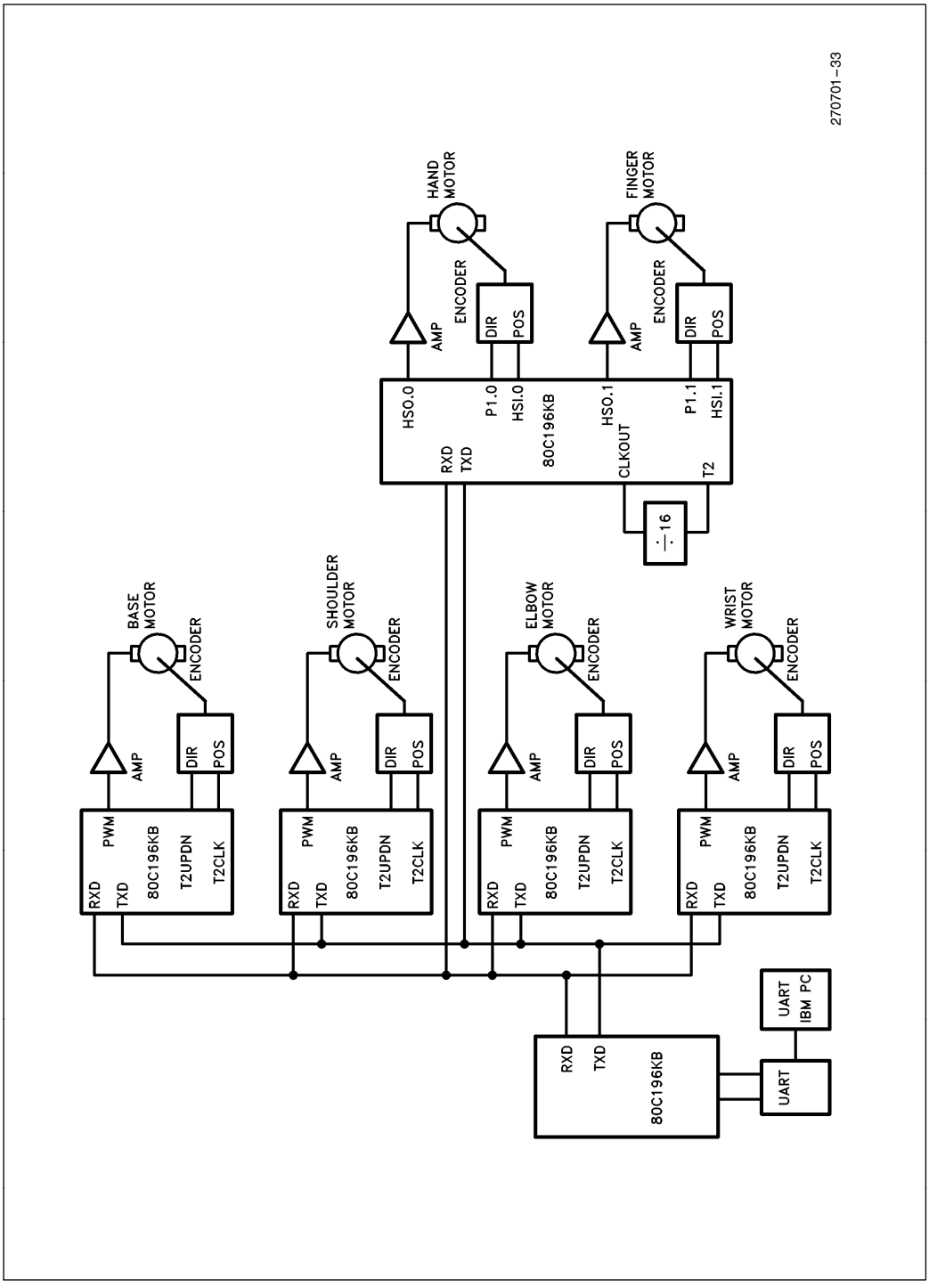



**Figure 32. Robot Control Hardware Block Diagram**

Switches on the robot were fed into 80C196KB I/O ports to provide a reference position when each motor starts up. Current sensing for each motor was fed back to the analog channels to provide an indication of any overrun or stress conditions. Limits were set for each motor to prevent the robot joints from entering positions where obstacles or mechanical limitations were reached.

Each motor was given a unique programming address for communication back to the master controller. The master controller sent commands with the address of whichever joint motor needed to be positioned or polled. The master 80C196KB communicated through a UART to the IBM-PC.

## 5.2 Human Interface

To control the robot, the human interface provided a variety of programming options.

The software features included:

Manual control via the keyboard
Editing robot command files
A Motor Control Command language
Table Display of motor position and status
Manual Programming mode
Table Positioning mode

The software front end developed only the basic features of robotic control to demonstrate the distributed control of servo motors.

## 5.3 Control Screen for the Robot

The screen for the control of the robot is shown in Figure 33. The screen displays a table of the position and status of each motor, shows the function keys used to execute commands or enter different modes and displays the keyboard keys for moving each robot joint up or down. The software has various modes for positioning and programming the robot.

## 5.4 Programmed Modes

The software provides for movement of the robot through table entry, execution of include command files or manually using the keyboard. The robot is positioned manually by entering the function key for manual mode and then pressing the predefined key for each joint motor to move up or down. As each key is released, a STOP command is issued to each motor. The motors are then polled and the current position updated in the table.

The table function allows for direct entry of the desired position and maximum velocity to position the motor when the table function key (F1) is pressed. After the

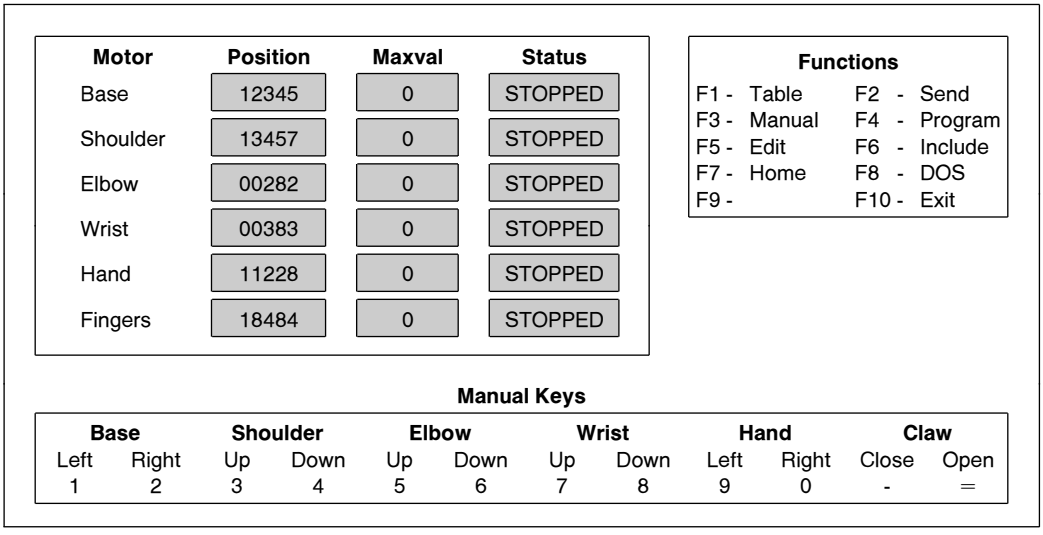



**Figure 33. Robot Control Screen**

key is pressed, individual positioning commands are sent to each motor. With maximum velocity set to zero, the motor is positioned using a position PID. A non-zero maximum velocity would position the motor using a velocity profile. The final method of positioning allowed for the sending of commands (MOTOR UP, MOTOR DOWN, STOP, POSITION or POLL) to each joint in the robot from an include file.

The include mode function key (F6) executes commands stored in a file. The command file can be entered using an external editor or using the on board editor, Turbo Prolog. A sample command file is shown in Figure 34. The command file allows for programming of the robot through a sequence of programmed tasks. The task of programming the robot is eased by a manual program mode.

The manual program mode generates a command file while manually positioning the robot. After pressing the program key (F4), the program mode is entered and the robot is moved by pressing the appropriate motion key for each joint motor. When the robot stops, the position of the robot is polled and translated into a position command and stored in a file. As the programmed task is executed, each position of the robot and the time delay between joint movements is recorded. When the task is complete, the file contains all the stored position commands necessary to execute the programmed task. The file can be edited with by entering the edit mode (F5) to fine tune the programmed task or execute the command file directly. The manual program, command file execution and editing modes allow for a variety of robotic tasks to be developed and tested easily.

## 6.0 CONCLUSION

Use of an 80C196KB in distributed control of servo motors has been demonstrated with the effective utilization of the onboard peripherals and high speed math capability of the 80C196KB. The high performance and integration of the 80C196KB minimized the hardware interface. The task of controlling the motor resided in the 80C196KB with the control algorithm residing in the master. With this approach, the centralized controller can be adapted to the performance requirements of the system.

Although not implemented, a learn mode could be added to the robot to provide programming using AI techniques. The IBM PC and Turbo Prolog software provided the demonstration vehicle for testing the control of the robot using distributed control. Use of artificial intelligence programming to position the robot could be incorporated with the Turbo Prolog package. The application of a vision system or a more complex control algorithm could be realized without modification to the hardware controlling the robot. A more cost effective solution is obtained by replacing the IBM-PC with one 80C196KB or 80C186 acting as a master controller.

Repetitive tasks programmed using the robot command language could be stored in tables in the master 80C196KB. The controller would send the stored commands to each motor and communicate, through a serial UART, to the rest of the manufacturing system. The master 80C196KB controller would then report status or receive commands. The choice of controller depends on the needs of the system. Distributed control of servo motors using the 80C196KB provides for maximum flexibility in the selection of the control algorithm without modification to the hardware control modules.

```
pos(3,4000,10)    ; move elbow to position 4000 with maximum velocity of 10
time(10)          ; delay 10 seconds
pos(1,1000,2)     ; move shoulder to position 1000 with maximum velocity of 2
time(20)          ; delay 20 seconds
pos(0,14000,5)    ; move base to position 14000 with maximum velocity of 5
```

**Figure 34. Sample Robot Command File**

## REFERENCES

1. Michael Brady, *Robot Motion: Planning and Control* (MIT Press, 1982).
2. C. S. G. Lee, *Tutorial on Robotics* (2nd edition) (IEEE Computer Society Press, 1989)
3. Electro Craft Corporation, "DC Motors Speed Controls Servo Systems", 1978.
4. Proceedings of Conferences on Applied Motion Control, University of Minnesota, 1986.

INTEL CORPORATION, 2200 Mission College Blvd., Santa Clara, CA 95052; Tel. (408) 765-8080

INTEL CORPORATION (U.K.) Ltd., Swindon, United Kingdom; Tel. (0793) 696 000

INTEL JAPAN k.k., Ibaraki-ken; Tel. 029747-8511

Printed in U.S.A./xxxx/0296/B10M/xx xx